Proceedings of the 2007 IEEE Symposium on Computational
Intelligence in Scheduling (CI-Sched 2007)

Global Estimations for Multiprocessor Job-Shop

Nodari Vakhania *Member, IEEE*[*]

Abstract. Classical job-shop scheduling problem (JSP) is one of the heaviest (strongly) NP-hard scheduling problems, which is very difficult to solve in practice. No approximation algorithms with a guaranteed performance exist. We deal with a natural generalization of this problem allowing parallel processors instead of each single processor in JSP, and an arbitrary task graph (without cycles) instead of a serial-parallel task graph in JSP. Parallel processors might be identical, uniform or unrelated. The whole feasible solution space grows drastically compared to JSP. However, as it turned out, parallel processors can also be used to reduce the solution space to a subspace, which is essentially smaller than even the corresponding solution space for JSP [1]. For large problem instances, this space still may remain too big. Here we propose different global estimations which allow us to reduce it further. By applying our bounds to the reduced solution space a class of exact and approximation algorithms are obtained. We are in the process of the implementation of our reduction algorithm and the bounds. Then we aim to carry out the experimental study comparing the behavior and the efficiency of the proposed bounds in practice.

1 Introduction

Classical job-shop scheduling problem (JSP) is one of the heaviest (strongly) NP-hard scheduling problems, which is very difficult to solve in practice. No approximation algorithms with a guaranteed performance exist. The problem is important because it reflects the actual operation in several industries, though it is still too restricted for many industries. We consider a natural generalization of this problem allowing parallel processors instead of each single processor in JSP, and an arbitrary task graph (without cycles) instead of a serial-parallel task graph in JSP. Parallel processors might be identical, uniform or unrelated. This meets better the needs of a vast amount of practical problems: A computer may have parallel processors each of which might be used by a program task, or in a manufacturing plant job might be allowed to be processed by any of the available parallel machines. Besides, the precedence relations might be more complicated than serial-parallel type relations. For example, the completion of two or more program tasks (subroutines) might be necessary before some other program task can be processed (as the latter task uses the output of the former tasks); this is a typical situation in parallel and distributed computations.

The whole feasible solution space grows drastically compared to JSP. However, as it turned out, parallel processors can also be used to reduce the solution space to a subspace, which is essentially smaller than even the corresponding solution space for JSP [1]. For large problem instances, this space may still remain too big. Here we propose different bounds which allow us to reduce it further. Combining our bounds with the already reduced solution space a class of exact and approximation algorithms can be obtained.

Our generalized problem is as follows. Given are the set of *tasks* or *operations*, $\mathcal{O} = \{1, 2, ..., n\}$ and m different processor groups. $\mathcal{M}_k$ is the kth group of parallel *processors* or *machines*, P_{kl} being the lth processor of this group. (A job in a factory, a program task in a computer or a lesson in a school are some examples of jobs. A machine in a factory, a processor in a computer, a teacher in a school are some examples of machines.) Each task should be performed by any processor of the given group. d_{iP} is the (uninterrupted) processing time of task i on processor P. Each group of parallel processors can be *unrelated*, *uniform* or *identical*. Unlike uniform machines which are characterized by an operation-independent speed function, unrelated machines have no uniform speed charac-

[*]Partially supported by CONACyT grant 48433. Science Faculty, State University of Morelos, Av. Universidad 1001, Cuernavaca 62210, Morelos, Mexico; fax: +52 777 329 70 40; e-mail: nodari@uaem.mx

teristic, i.e., a machine speed is operation-dependent; that is, processing times d_{iP} are independent, arbitrary integer numbers. In case of identical machines task processing times are processor-independent, all processors have the same speed.

We have the *resource constraints*: For each two jobs i, j such that $P(i) = P(j) = P$, either $s_i + d_{iP} \leq s_j$ or $s_j + d_{jP} \leq s_i$ should hold, where s_i is the starting time of i and $P(i)$ is the processor to which task i is assigned; in other words, any processor can handle only one task at a time. The *precedence constraints* are as follows. For each $i \in \mathcal{O}$ we are given the set of immediate predecessors $pred(i)$ of task i, so that i cannot start before all tasks from $pred(i)$ are finished. Task i becomes *ready* when all tasks from $pred(i)$ are finished.

A *schedule (solution)* is a function which assigns to each task a particular processor and a starting time (on that processor). A *feasible schedule* is a schedule satisfying above constraints. An *optimal schedule* is a feasible schedule which minimizes the *makespan*, that is, the maximal task completion time. As it is well-known, an optimal schedule of JSP is among so-called *active schedules*: in an active schedule no operation can start earlier than it is scheduled without delaying some other operation (for example see Lageweg, Lenstra and Rinnooy Kan [2] for the details).

Applying commonly used notation for scheduling problems, we use $J||C_{\max}$, $JR|prec|C_{\max}$, $JQ|prec|C_{\max}$ and $JP|prec|C_{\max}$, respectively to denote JSP and the versions of our generalized problem with unrelated, uniform and identical processors, respectively. If in an instance of our generalized problem from each group of processors all processors except an arbitrarily selected one is eliminated, then a corresponding instance of JSP is obtained. JSP and hence the generalized problem are strongly NP-hard: though the construction of each feasible schedule takes a polynomial (in the number of operations and machines) time, for finding an optimal schedule we might be forced to enumerate an exponential number of feasible schedules. Since each feasible schedule can be rapidly generated, different heuristics or priority dispatching rules are used for a rapid generation of some feasible schedule(s). The simplest considerations which reflect priority dispatching rules are not enough to obtain a solution with a desirable good quality. If the quality of the required solution is important, we need to work with a larger subsets of feasible solution space to guarantee the optimality.

An algorithm, reducing the number of all feasible solutions of the generalized problem was proposed in Vakhania & Shchepin [1]. Surprisingly, with the probability of almost 1, the number of feasible solutions generated by this algorithm, as compared to the number of all active feasible schedules, *decreases* with the number of machines and operations in each group of machines and operations, as follows. If we let ν and μ to be the number of operations and machines in each subset of operations and machines, then with a probability of almost 1, the algorithm generates approximately $(\mu)^{m\nu}$ and $2^{m(\mu-1)}\mu^{m\nu}$ times less feasible schedules than the number of all active feasible schedules of any corresponding instance of JSP and our generalized problem, respectively. This algorithm may still generate an inadmissible number of feasible schedules for large real-life problem instances. Branch-and-bound algorithms, as well as some approximation algorithms, such as beam search, incorporate (lower) bounds for the further reduction of the solution space. In this paper, we suggest different bounds for the three versions of our generalized problem. By incorporating these bounds with the reduced solution tree generated by the algorithm in [1], different branch-and-bound and approximation algorithms are obtained. Currently, we are in the process of the implementation of our reduction algorithm and our bounds. Then we aim to carry out the experimental study comparing the behavior and the efficiency of the proposed bounds in practice. We hope we may have some preliminary results by the conference date.

A brief overview of some related literature is as follows. The earliest work mentioning a generalization of JSP is that of Giffer & Thompson [3] in which, instead of unrelated processors and arbitrary precedence relations in our generalized problem, identical processors and serial-parallel type precedence relations were considered. More recently, an extension of JSP with general multi-purpose machines was studied by Brucker & Schlie [4], Brucker, Jurisch & Kramer [5], Vakhania [6], and a lower bound for the special case of this problem when an operation processing time is a constant (i.e. machine-independent) was suggested by Jurisch [7]. Shmoys, Stein & Wein [8] have proposed a polynomial approximation randomized algorithm for $R|chain|C_{\max}$ which can be applied for the version of our problem with serial-parallel precedence relations $JR|serial|C_{\max}$. Dauzére-Péres & Pauli [9] has proposed a Tubu-search algorithm and Vakhania [10] has suggested a version of beam search algorithm for the generalized problem. Ivens & Lambrecht [11], Shutten [12] study different extensions of JSP including extensions with setup and transportation times.

2 Basic Concepts

The algorithm in [1] generates the so-called *compact solution tree*. We denote this tree by T and call its node a *stage*. A complete schedule from the reduced feasible solution space corresponds to each path from the root to a leaf in T, each intermediate stage represents a partial solution. The (partial or complete) solution, corresponding to stage h is denoted by σ_h. We denote a bunch of concurrent ready tasks at each stage h, the candidates determined by the algorithm [1] to be scheduled at that stage, by $\mathcal{C}_h$. We branch in T at stage h resolving the resource (machine) conflicts in $\mathcal{C}_h$. One immediate successor of node h is generated for each task of $\mathcal{C}_h$. Assume h' is the immediate successor of h with task $i \in \mathcal{C}_h$ scheduled at that stage. Then with the arc (h, h') two labels are associated: the task i and the processor on which this task is actually scheduled. In this way, there are generated $|\mathcal{C}_h|$ extensions of the current partial schedule σ_h in our branch-and-bound tree T. σ_h can be clearly seen as a (partial) permutation of n tasks. For $i \in \sigma_h$ in that permutation, we shall use the upper index for specifying the particular processor on which task i is scheduled in σ_h. In particular, $\sigma_h i^P$ is an extension of σ_h with task i scheduled on processor P. Note that the relative order of two arbitrary tasks $i, j \in \sigma_h$ is relevant only if $P(i) = P(j)$.

We denote by $\mathcal{O}_k$ the set of tasks to be performed on a processor of kth group and by $\mathcal{O}_{kh}$ the subset of tasks from $\mathcal{O}_k$ not yet scheduled by stage h. Each feasible solution σ_h is represented by a directed weighted graph G_h. The digraph $G_0 = (X, E_0)$ we associate with the root of T. To each task $i \in O$ corresponds the unique node $i \in X$. There is one fictitious initial node 0, preceding all nodes, and one fictitious terminal node $n + 1$, succeeding all nodes in G_0. E_0 is the arc set consisting of the arcs (i, j), for each task i, directly preceding task j; $(0, i) \in E_0$ if task i has no predecessors and $(j, n + 1) \in E_0$ if task j has no successors. We denote by $w(i, j)$ the weight associated with $(i, j) \in E_0$; initially, we assign to $w(i, j)$ the minimal processing time of task i, later we correct these weights when we assign a task to the particular processor. Let (h, h') be an edge in T with task j scheduled at iteration h' on processor P. Then we obtain $G_{\sigma_{h'}}$ from G_{σ_h} as follows. We complete the arc set of the latter graph with the arcs of the form (i, j), with the associated weights $w(i, j) = d_{iP}$, for each task i, scheduled earlier on the processor P. We correct the weights of all arcs incident out from node j $(j, o) \in E_0$, as $w(j, o) := d_{jP}$. It is easily seen that the length of a critical path in $G_{h'}$ is the makespan of the (partial or complete) solution $\sigma_{h'} = \sigma_h j^P$ which we denote by $|\sigma_{h'})|$; by $\tau_h(i)$ we denote the length of a longest path to node i in G_h, that is, the earliest possible starting time of task i at stage h.

Since the critical path length from node 0 to a node o in G_h is a lower bound on the starting time of operation o in schedule σ_h and in any its successor schedule, we call it the *early starting time* or the *head* of operation o by stage h and denote by $\text{head}_h(o)$. $R_h(M)$ is the *release time* of machine M at stage h, that is, the completion time of the operation, scheduled last by that stage on M.

Now we derive auxiliary multiprocessor scheduling problem which we shall use for calculation of our lower bounds. Remind that if a lower bound $L(\sigma_h)$ of the partial solution σ_h is more than or equal to the makespan $|\sigma|$ of some already generated complete solution σ, then all extensions of σ_h can be abandoned. It is clear that $L(\sigma_h)$ cannot be greater than the makespan of the best potential extension of σ_h (since then we could loose this extension), but it should be as close as possible to this value (because then the more are the chances that $L(\sigma_h) \geq |\sigma|$).

Let us first note that a trivial lower bound $L_T(\sigma_h o^Q)$ for the partial solution $\sigma_h o^Q$ of an instance of our generalized problem can be obtained as follows. For σ_h and $o \in \mathcal{C}_h$, let $L_T(\sigma_h o^Q) = \tau(\sigma_h) + \text{tail}_h(o)$, where $\text{tail}_h(o)$, called the *tail* of operation o at stage h, is the critical path length from a direct successor-node of o to the sink node of G_h. Evidently, this bound ignores all yet unresolved potential conflicts, i.e. the processing times of yet unscheduled tasks. Though it is easy and fast to obtain L_T, it is clear that we cannot get a good estimation of the desired optimal makespan by the complete ignorance of the potential contribution of all unscheduled tasks. A stronger lower bound would take into account a possible contribution of the latter tasks (this would obviously need additional computational efforts). Clearly, we cannot know in advance how yet unresolved conflicts will be resolved in an optimal schedule. But we can make some assumptions about this ("simulating" in advance some "future" resource constraints). However, we should be careful since we are not allowed to violate the condition $L(\sigma_h) \leq |\sigma'|$, σ' being an arbitrary complete extension of σ_h. Roughly speaking, we need to make an optimal assumption about how the future resource conflicts will be resolved; this will involve some optimal scheduling on parallel machines.

For JSP, most commonly is used a one-machine relaxation (for example see [13], [14], [16], [2], [18]): all resource constraints are relaxed (ignored) except the ones of a one particular (not yet completely scheduled) machine, and the resulted one-machine problem with heads and tails, $1|r_i, q_i|C_{max}$ is then solved. A *bottleneck* machine is a one which results the maximal makespan among all yet unscheduled machines (intuitively, a bottleneck machine gives a maximal expected contribution in the makespan of extensions of σ_h). This approach can be generalized as follows. Basically, we relax the resource constraints on all machines except the ones from some (bottleneck) set of machines $\mathcal{M}_k$. To be specific, let at iteration h, $|\mathcal{O}_{kh}| \geq 2$, i.e., there are unresolved resource constraints associated with the machines of $\mathcal{M}_k$. An operation $i \in \mathcal{O}_{kh}$ is characterized by its early starting (release) time $\text{head}_h(i)$ and tail $\text{tail}_h(i)$; that is, i cannot be started earlier than at time $\text{head}_h(i)$, and once it is completed, it will take at least $\text{tail}_h(i)$ time for all successors of i to be finished. i can be scheduled on any of the machines of $\mathcal{M}_k$ and has a processing time d_{iP} on machine $P \in \mathcal{M}_k$. Each machine $P \in \mathcal{M}_k$ has its release time $R_h(P)$. Observe that the operation tails and release times are derived from G_h (this ignores all unresolved by stage h resource constraints). Besides, the tails require no machine time, i.e., time on any of the machines of $\mathcal{M}_k$. We are looking for an optimal (i.e., minimizing the makespan with tails) ordering of the operations of $\mathcal{O}_{kh}$ on machines from $\mathcal{M}_k$ under the above stated conditions. Thus for each stage h for the partial solution σ_h we obtain the auxiliary problem of scheduling tasks with release times and tails on a group of parallel machines $\mathcal{M}_k$ with the objective to minimize the makespan. Let us denote this auxiliary problem by $\mathcal{A}_{kh}$ and the respective optimal makespan by $|\mathcal{A}_{kh}|$.

Let μ_h be the set of indexes of all machine groups such that for each $k \in \mu_h$, $|\mathcal{O}_{kh}| \geq 2$. It is clear that $|\mathcal{A}_{kh}|$, for any $k \in \mu_h$, is a lower bound for node h. We may find all $|\mu_k| \leq m$ lower bounds for node h and take the maximum thus finding a bottleneck machine group. Thus instead of dealing with $1|r_i, q_i|C_{max}$ in case of JSP, now we deal with $R|r_i, q_i|C_{max}$. Both problems are NP-hard, though there exist exponential algorithms with a good practical behavior for the first above problem, have been commonly used in one-machine relaxation based branch-and-bound algorithms for JSP (see, for example McMahon & Florian [18], Carlier [15] and Carlier & Pinson [16]). Unfortunately, there are no known algorithms with good practical performance for $P|r_i, q_i|C_{max}$ (the version with identical machines) and so much for $R|r_i, q_i|C_{max}$. In the following section we suggest several ways to obtain strong lower bounds for these problems.

3 Lower bounds

We first describe lower bounds based on earlier existing algorithms. Then we suggest alternative lower bounds. Carlier & Pinson [17] have suggested a lower bound for $JP|prec|C_{max}$. They proposed an $O(n \log n + nm \log m)$ algorithm for non-sequential version of $P|r_i, q_i, prmt|C_{max}$ which is a tight lower estimation of the optimal makespan for $P|r_i, q_i, prmt|C_{max}$. At the expense of weakening the bound, the solution of the above problem can be used as a lower bound for the version with unrelated machines as we describe below.

Let $d_o^{\min}$ be the minimal processing time of operation $o \in \mathcal{O}_k$, i.e., $d_o^{\min} = \min\{d_{oM}, M \in \mathcal{M}_k\}$. We replace the unrelated machine group $\mathcal{M}_k$ with the identical machine group $\mathcal{M}_k^{'}$, defined as follows: the number of machines in both groups is the same, and for each $o \in O_k$ and $M \in \mathcal{M}_k^{'}$, $d_{oM} = d_o^{\min}$. It is clear that an optimal solution of the obtained instance of $P|r_i, q_i, prmt|C_{max}$ with $\mathcal{M}_k^{'}$ is no more than that of the corresponding instance of $R|r_i, q_i, pmtn|C_{max}$ with $\mathcal{M}_k$. Hence, the former solution can be used for the calculation of a lower bound for the original problem. Obviously, the bound obtained in this way would be weak if the difference between the above two solutions is significant. It might be possible to find a better "approximation" with an identical machine group of the unrelated machine group $\mathcal{M}_k$, i.e., to increase d_{oM}, $o \in \mathcal{O}_k$, $M \in \mathcal{M}_k^{'}$ (this could be the subject of a further research).

For uniform machines, we can obtain a stronger lower bound by using the algorithm of Federgruen & Groenevelt [21] for the problem $Qm|r_i, q_i, pmtn|C_{max}$ with the time complexity of $O(tn^3)$ (here t is the number of machines with distinct speeds).

As to $JR|prmt|C_{max}$, the technique based on linear programming of Lawler & Labetoulle [24] yields a polynomial-time algorithm for $Rm|r_i, q_i, pmtn|C_{max}$. This is clearly a lower estimation of the optimal makespan for $Rm|r_i, q_i|C_{max}$ which, in turn, provides a lower bound for $JR|prmt|C_{max}$.

Now we describe alternative methods to obtain lower bounds. For the versions with identical and uniform machines, our lower bounds are obtained in an almost linear (in $|\mathcal{O}_{kh}|$ and $|\mathcal{M}_k|$) time. For the version with unrelated machines, we apply linear programming. We obtain a lower estimation, which is not a strict lower bound for the same version again in almost linear time. This bound can be used in approximation algorithms such is a beam search.

For simplifying the notations, let $a_i = \text{head}_h(i)$ and

$q_i = \mathrm{tail}_h(i)$, for $i \in \mathcal{O}_{kh}$, where $k \in \mu_h$. Let, further, d_i^S be the processing time of i in S (d_i^S may vary from schedule to schedule depending on the particular machine, to which i is assigned), t_i^S ($c_i^S = t_i^S + d_i^S$, respectively), be the starting (finishing, respectively) time of operation i in schedule S. We call $c_i^S + q_i$ the *full* completion time of operation i.

First we apply a version of a "greatest tail heuristic" to the operations of $\mathcal{O}_{kh}$: iteratively, among all ready operations, we determine a one with a longest tail and schedule it on a machine on which the minimal completion time of this operation is reached. In the formal description below we refer to such a machine as a *quick machine*).

```
ALGORITHM GREATEST_TAIL
BEGIN
    (0)  $t := \min\{a_i | i \in \mathcal{O}_{kh}\}$;  $A := \mathcal{O}_{kh}$;
         $R_P := \max\{R_h(P), t\}$, for all $P \in \mathcal{M}_k$; { $R_P$ is the release of machine $P$ }
    (1)  Among the unscheduled jobs $l \subset A$ with $a_l \le t$, schedule next job $j$ with the greatest tail on its quick machine $Q$ (break ties by selecting a quick machine with the minimal release time);
         $t_j := \max\{t, R_Q\}$;  $R_Q := t_j + d_{jQ}$;  $A := A \setminus \{j\}$;
         IF $A \neq \emptyset$ THEN $t := \max\{\min\{R_P | P \in \mathcal{M}_k\}, \min\{a_i | i \in A\}\}$; GOTO (1)
             ELSE RETURN $\{t_j\}$, $j = 1, 2, ..., n$;
END.
```

It is easily seen that the time complexity of the above algorithm is $O(\mu n \log n)$, where $\mu = |\mathcal{M}_k|$. In the following, S denotes a greatest tail schedule obtained by the algorithm GREATEST_TAIL for the operations of $\mathcal{O}_{kh}$. S, in general, consists of a number of *blocks*. Intuitively, a block is a maximal independent part in a schedule. More precisely, B is a maximal consecutive part in S (that is, a maximal sequence of the successively scheduled jobs on the adjacent machines), such that for each two successively scheduled tasks i and j, task j starts no later than task i finishes. Let $r \in \mathcal{O}_{kh}$ be the latest scheduled in S operation such that $c_r^S + q_r$ equal to $|S|$ (clearly, there exists at least one such operation in S). If $t_r = a_r$, S is optimal (as task r is scheduled on its quick machine) and $|S| = |\mathcal{A}_{kh}|$ is the optimal makespan. If $t_r > a_r$, then r potentially might be completed earlier by rescheduling some operation(s), scheduled before r, after r. Next we will see how this works.

Let us call an operation $l \in S$, scheduled before r with $q_l < q_r$, an *emerging* operation in S, if l belongs to the same block as r. The set of operations scheduled in S between the latest scheduled emerging operation and operation r is called the *kernel*. Thus any kernel operation has a tail, no less than q_r. We increase "artificially" the readiness time of some emerging operation l by setting $a_l := a_r$ and apply again algorithm GREATEST_TAIL. Then we will get a new greatest tail schedule, S_l, in which l is rescheduled after all operations of kernel. We call the above rescheduling of task l its *application*. Once we apply l, we liberate space for kernel operations (in particular, for operation r). These operations will be rescheduled earlier in the new obtained greatest tail schedule S_l. Hence, the makespan in S_l might be decreased (in comparison with that in S). Let us call the maximal magnitude, by which in this way a kernel operation can be rescheduled earlier, the *shifting value* of that operation. Note that it makes no sense to apply any non-emerging operation. For the further details, we refer the reader to Vakhania [19] and [20].

Theorem 1 *The shifting value of any kernel operation, including r, is strictly less than the maximal operation processing time d_{max}.*

Proof. Indeed, let an emerging operation l be applied in S and let i be the earliest kernel operation, started in the time interval $[t_l^S, c_l^S]$ in S_l after the rescheduling of l. Since $q_i > q_l$, $a_i > t_l$ as otherwise i, instead of l, would be scheduled at the moment t_l in S. Therefore, $t_i^{S_l} > t_l^S$, i.e., i is antedated by a gap in S_l. Besides, the delay of i in S (i.e., $t_i^S - a_i$) can be only less than d_l^S. Suppose there exists another emerging operation in S_l and we apply it; then similarly, we will get another kernel operation antedated by a gap in the resulted schedule. After the application of μ', $\mu' \le |\mathcal{M}_k|$ emerging operations, μ' kernel operations will be started in the time interval, occupied earlier by some emerging operation, and each of these kernel operations will be antedated by a gap. The shifting value of each succeeding kernel operation is no more than that of the first μ' kernel operations. Hence, the shifting value of any kernel operation is strictly less than the maximal processing time of rescheduled emerging operations.$\diamondsuit$

Thus the successive application of no more than $|\mathcal{M}_k|$ emerging operations is sufficient to construct a greatest tail schedule, say S', in which the first $|\mathcal{M}_k|$ kernel operations (all kernel operations if their number is less than $|\mathcal{M}_k|$) are antedated by the newly arisen gaps. Let C be the sequence of kernel operations in S' (observe that C starts with the kernel operations in S', antedated by the newly arisen gaps). In S', an operation

$i \in C$ either starts at time a_i or it starts right at the moment of completion of another operation of C. Hence, $\min\{t_i^{S'} \mid i \in C\} = \min\{a_i \mid i \in C\}$ is the minimal possible starting time for C.

Let C^S be the sequence in which the kernel operations were scheduled in S. Observe that although C^S might be different from C, all the applied in S' emerging operations have been initially scheduled before C^S in S. In S, the sequence C^S is started with a delay which is determined by the finishing times of the μ' emerging operations directly preceding kernel operations in S. Suppose that, respecting this delay of C^S in S, the sequence C itself is optimal (i.e. it minimizes the maximal completion time of kernel operations, subject to the release times of the $\mathcal{M}_k$ machines). Then from the definition of C^S and r, and the earlier made observation, $|S| - d_{max} = c_r^S + q_r^S - d_{max}$ is a lower bound on the optimal schedule makespan. Note that its calculation takes $O(\mu n \log n)$ time. This bound, in general is not a strict lower bound for $JP|prec|C_{max}$ (as the sequence C^S is not optimal), though it can be successfully applied as a thorough estimation in approximate algorithms such as beam search (see for example [10]).

The above bound can be easily transferred to a strict lower bound for the versions with identical and uniform machines. In principal, we need to find a good lower estimation for an optimal sequence of kernel operations. This task can be solved in almost linear time for both, identical and uniform machines, while for unrelated machines we will apply (also polynomial) linear programming. We obtain a good lower estimations for the problem $Q|r_i|C_{max}$ (which itself is NP-hard) by solving its preemptive version $Q|r_i, pmtn|C_{max}$ in $O(n \log n + mn)$ time (see Sahni & Cho [25] and Labetoulle, Lawler, Lenstra & Rinnooy-Kan [23]). If we ignore the operation release times (this, in general, is possible since the optimal makespan without the release times is no more than that with the readiness times), we can apply an $O(n + m \log m)$ algorithm for $Q|pmtn|C_{max}$ by Gonzalez & Sahni [22]. Similarly, we obtain a good lower estimation for $R|r_i|C_{max}$ by solving its preemptive version by linear programming (see Lawler & Labetoulle [24]).

The above estimations provide us with the earliest possible finishing time, c^*, of the kernel operations. Let $q = \min\{q_i, i \in C^S\}$ and d be the maximal processing time among all emerging operations in S. Then $L_1(C^S) = c^* + q - d$ is clearly a lower bound on the makespan of $\mathcal{A}_{kh}$. This bound can be further strengthen. Earlier we saw how the sequence C is obtained after the application of no more than $|\mathcal{M}_k|$ emerging operations (which were the latest scheduled ones in S). Denote this set of emerging operations by E and the set of all emerging operations in S by $\mathcal{E}$. In general, emerging operations from $\mathcal{E} \setminus E$ can be applied instead of some emerging operations of E (note that emerging operations from the latter set precede those from E in S). Indeed, if emerging operations $l_1, ..., l_p$ are all released by time t, they will be successively scheduled in S till the moment when the earliest non-emerging operation gets ready. Thus we may have a choice, which emerging operations to apply. By choosing emerging operations from E we guarantee that the sequence C will start without any delay; at the same time, the rescheduled after C emerging operations of E having "long enough" tail may obviously affect the resulted makespan (i.e. the maximal *full* job completion time.

This consideration makes it clear that by taking into the account the actual tails and processing times of the rescheduled emerging operations, the earlier bound might be further improved. A simple solution might be as follows. Assume on each machine from $\mathcal{M}_k$ an operation of C is scheduled (otherwise, as it is easily seen, there is no need in this additional estimation). At least one emerging operation should be rescheduled after C; hence, any $E' \subset \mathcal{E}$ will be fully completed no earlier than at time $L_2(E') = c' + d' + q'$, where c' is the minimal finishing time of the operations of C scheduled last on one of the machines of $\mathcal{M}_k$, $d' = \min\{d_{iP}, i \in \mathcal{E}, P \in \mathcal{M}_k\}$ and $q' = \min\{q_i, i \in \mathcal{E}\}$. Thus $L_{kh} = \max\{L_1(C), L_2(E)\}$ is a lower bound for $\mathcal{A}_{kh}$.

4 Conclusions

Some of the developed global estimations are strict lower bounds and the others not. The latter bounds, which were obtained in almost linear time, can be used in any solution tree based approximation algorithm, such as, for example, beam search. Similarly, applying our strict lower bounds in T, we obtain exact branch-and-bound algorithms. Once implemented and tested, the above algorithms can be used as an algorithmic engine for a decision support system for generalized job shop scheduling problems. The presented algorithms can be easily aggregated by an additional graph-completion mechanism for taking into the account the transportation times and other real-life job shop scheduling problems [11] and [12].

References

[1] N. Vakhania and E. Shchepin. "Concurrent operations can be parallelized in scheduling multiprocessor job shop". *Journal of Scheduling* 5, p.227-245 (2002).

[2] B. J. Lageweg, Lenstra J.K., Rinnooy Kan A.H.G., "Job Shop Scheduling by Implicit Enumeration", *Management Science* 24,441-450 (1977).

[3] B. Giffer and G.L.Thompson, "Algorithm for Solving Production Scheduling Problems", *Oper. Res.* 8,487-503 (1960).

[4] P. Brucker and R. Schlie. "Job shop scheduling with multi-purpose machines". *Computing*, 45, 369-375 (1990).

[5] P. Brucker, B.Jurisch and A.Kramer, "Complexity of scheduling problems with multi-purpose machines", *Annals of Operations Research"*, 70, 57-73 (1997).

[6] N.N. Vakhania, "Assignment of jobs to parallel computers of different throughput", *Automation and Remote Control*, 56, 280-286 (1995).

[7] Jurisch B., "Lower bounds for job-shop scheduling problem on multi-purpose machines", *Discrete Applied Mathematics* 58, 145-156 (1995).

[8] D.B. Shmoys, C.Stein and J.Wein, "Improved approximation algorithms for shop scheduling problems", *SIAM J. on Computing* 23, 617-632 (1994).

[9] Dauzére-Pérés and J.Paulli. "An integrated approach for modeling and solving the general multiprocessor job shop scheduling problem with tabu search". *Annals of Operations Research*, 70, 281-306 (1997).

[10] N. Vakhania, "Global and local search for scheduling job shop with parallel machines", *Lecture Notes in Artificial Intelligence (IBERAMIA-SBIA 2000)*, 1952, p.63-75 (2000).

[11] P. Ivens and M. Lambrecht, "Extending the shifting bottleneck procedure to real-life applications", *European J. of Operational Research* 90, 252-268 (1996).

[12] J.M. Schutten, "Practical job shop scheduling", *Annals of Operations Research* 83, 161-177 (1998).

[13] J. Adams, E.Balas and D.Zawack, "The Shifting Bottleneck Procedure for Job Shop Scheduling", *Management Science*, 34, 391-401 (1988).

[14] J. Blazewicz, W. Cellary, R.Slowinski and J. Weglarz, "Scheduling under resource constraints - Deterministic models". *Annals of Operations Research*, 7 (1986)

[15] J. Carlier, "The one-machine sequencing problem", *European J. of Operational Research"* 11, 42-47 (1982).

[16] J. Carlier and E.Pinson, "An Algorithm for Solving Job Shop Problem", *Management Science*, 35, 164-176 (1989).

[17] J. Carlier and E.Pinson, "Jakson's pseudo preemptive schedule for the $Pm/r_i, q_i/C_{max}$ problem", *Annals of Operations Research* 83, 41-58 (1998).

[18] G.B. McMahon and M. Florian, "On scheduling with ready times and due dates to minimize maximum lateness", *Operations Research* 23, 475-482 (1975).

[19] Vakhania N. Scheduling equal-length jobs with delivery times on identical processors. *Int. J. Computer Math.*, 82 (2002).

[20] N. Vakhania. "A better algorithm for sequencing with release and delivery times on identical processors". *Journal of Algorithms* 48, p.273-293, 2003.

[21] A. Federgruen and H. Groenevelt, "Preemptive scheduling of uniform machines by ordinary network flow techniques", *Management Science* 32, 341-349 (1986).

[22] T. Gonzalez and S. Sahni, "Preemptive scheduling of uniform processor systems", *Journal of the ACM* 25, 92-101 (1978).

[23] J. Labetoulle, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy-Kan, "Preemptive scheduling of uniform machines subject to release dates", in *Pulleyblank*, 245-261 (1984).

[24] E.L. Lawler and J.Labetoulle, "On preemptive scheduling of unrelated parallel processors by linear programming", *J. of the ACM* 25, 612-619 (1978).

[25] S. Sahni and Y.Cho, "Nearly on-line scheduling of a uniform processor system with release times", *SIAM J. on Computing* 8, 275-285 (1979).